NIST Interagency Report 7802
(DRAFT)

National Institute of Standards and Technology
U.S. Department of Commerce

Trust Model for Security Automation Data 1.0 (TMSAD) (DRAFT)

Harold Booth
Adam Halbardier

NIST Interagency Report 7802 (DRAFT)

Trust Model for Security Automation Data 1.0 (TMSAD) (DRAFT)

Harold Booth
Adam Halbardier

C O M P U T E R   S E C U R I T Y

Computer Security Division
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8930

July 2011

U.S. Department of Commerce

Gary Locke, Secretary

National Institute of Standards and Technology

Dr. Patrick D. Gallagher, Director

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analysis to advance the development and productive use of information technology. ITL's responsibilities include the development of technical, physical, administrative, and management standards and guidelines for the cost-effective security and privacy of sensitive unclassified information in Federal computer systems. This Interagency Report discusses ITL's research, guidance, and outreach efforts in computer security and its collaborative activities with industry, government, and academic organizations.

National Institute of Standards and Technology Interagency Report 7802 (DRAFT)
25 pages (July 2011)

Acknowledgments

The authors wish to thank their colleagues who reviewed drafts of this document and contributed to its technical content.

Abstract

This report defines the Trust Model for Security Automation Data 1.0 (TMSAD), which permits users to establish integrity, authentication, and traceability for security automation data. Since security automation data is primarily stored and exchanged using Extensible Markup Language (XML) documents, the focus of the trust model is on the processing of XML documents. The trust model is composed of recommendations on how to use existing specifications to represent signatures, hashes, key information, and identity information in the context of an XML document within the security automation domain.

Audience

The primary audiences for the TMSAD specification are developers of security automation specifications, IT products that could leverage TMSAD's capabilities, and organizations that could take advantage of TMSAD to establish integrity, authentication, and traceability of their security automation data. NIST welcomes feedback on improving the TMSAD specification.

Trademark Information

Table of Contents

List of Appendices

List of Figures and Tables

1. Introduction

A trust model is a necessary component for handling security automation data to permit users to establish integrity, authentication, and traceability for the data. The trust model can be leveraged to determine authorization—that a requestor of a particular piece of information is permitted access to that information, or that a particular piece of content is permitted to be processed. A trust model may also be used to implement traceability of results, giving increased assurance that a set of results are from a particular source. Finally, a trust model will allow for content integrity to be affirmed, assuring that content has not been modified since it was produced, whether by human or machine.

1.1 Purpose and Scope

This document provides guidelines and recommendations for how a common trust model, called the Trust Model for Security Automation Data (TMSAD), can be applied to specifications within the security automation domain, such as Security Content Automation Protocol (SCAP). Since information in the security automation domain is primarily exchanged using Extensible Markup Language (XML), the focus of this model is on the processing of XML documents [XML]. The trust model is composed of recommendations on how to use existing specifications to represent signatures, hashes, key information, and identity information in the context of an XML document within the security automation domain.

This document makes extensive use of the W3C recommendation *XML Signature Syntax and Processing* [XMLDSIG], referencing the features and syntax of [XMLDSIG]. The requirements of those features are described in the W3C recommendation and are not repeated in this document. It is expected that readers of this document will already be familiar with the details of [XMLDSIG].

Detailing a method for managing and exchanging public keys is out of scope for this document. This document provides information on how X.509 certificates or public keys may be represented within the model; however, this document defers to the content consumer for establishing a trust relationship to a particular identity or key.

1.2 Document Structure

This report is organized into the following major sections:

- Section 2 defines selected terms and abbreviations used in this specification.
- Section 3 provides an overview of related specifications and standards.
- Section 4 defines the high-level conformance rules for this specification.
- Section 5 defines the cryptographic algorithms and parameters to those algorithms that may be used for hashing and signing.
- Section 6 provides a brief overview of the *XML Signature Syntax and Processing* specification; it defines how that specification will be used and what additional requirements security automation will impose.
- Section 7 describes processing requirements for the trust model.
- Appendix A provides some examples of usage of the defined trust model.
- Appendix B lists normative and informative references.

- Appendix C provides a change log that documents significant changes to major drafts of the specification.

1.3 Document Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

Text intended to represent computing system input, output, or algorithmic processing is presented in `fixed-width Courier font`.

Table 1 shows the conventional XML mappings used in this document.

Table 1 – Conventional XML Mappings

Prefix	Namespace	Schema
dc	http://purl.org/dc/elements/1.1/	Simple Dublin Core elements
dsig	http://www.w3.org/2000/09/xmldsig#	Interoperable XML digital signatures
dt	http://scap.nist.gov/schema/xml-dsig/1.0	Trust Model for Security Automation Data extensions
xs	http://www.w3.org/2001/XMLSchema	XML Schema schema document

2. Abbreviations

This section defines selected abbreviations, including acronyms, used within the document.

DSS – Digital Signature Standard
ECDSA – Elliptic Curve Digital Signature Algorithm
FIPS – Federal Information Processing Standards
IR – Interagency Report
RFC – Request for Comments
SCAP – Security Content Automation Protocol
SHA – Secure Hash Algorithm
SP – Special Publication
TMSAD – Trust Model for Security Automation Data
URI – Uniform Resource Identifier
W3C – World Wide Web Consortium
XML – Extensible Markup Language
XSLT – Extensible Stylesheet Language Transformation

3. Relationship to Existing Specifications and Standards

This document makes use of existing specifications such as *XML Signature Syntax and Processing* [XMLDSIG] to establish a trust model. This document further specifies and constrains usage of [XMLDSIG] and other W3C recommendations to satisfy requirements exposed within the security automation domain.

Although *XML Signature Syntax and Processing Version 1.1* [XMLDSIG-11] is not a W3C recommendation as of mid-2011, this document adds requirements for selected cryptographic algorithms consistent with the requirements currently included in [XMLDSIG-11].

4. Conformance

Products and organizations may want to claim conformance with this specification for a variety of reasons. For example, a software vendor may want to assert that its product uses the trust model properly and can interoperate with any other product using the trust model. Another example is a policy mandating that an organization use the trust model for establishing suitability of content for use, or establishing provenance of content.

This section provides the high-level requirements that a product or document containing signature information MUST meet for conformance with this specification. Most of the requirements listed in this section reference other sections in the document that fully define the requirements.

Other specifications that use the trust model defined within this document MAY define additional requirements and recommendations. In addition, other specifications or standards MAY define additional requirements on the correct implementation of the cryptographic algorithms in specific environments or situations. Such requirements and recommendations are outside the scope of this publication.

4.1 Product Conformance

There are two types of products that may be conformant with the trust model: content authors and content consumers. Content authors are products that generate content that uses the trust model, while content consumers are products that process content that leverages the trust model. All products claiming conformance with this specification MUST comply with the following requirements:

1. Content consumers MUST consume and correctly process well-formed trust model documents as defined in Section 6. This includes following all of the processes defined in Section 7.

2. Content authors MUST ensure that all trust model documents they produce are well-formed. This includes following all of the processes defined in Section 7, and adhering to the syntax, structural, and other trust model document development requirements defined in Section 6.

3. All products MUST support the algorithms and parameters identified in Section 5.

4. Make an explicit claim of conformance to this specification in documentation provided to end users.

4.2 Content Conformance

Organizations creating or maintaining documents that claim conformance with this specification SHALL adhere to the syntax, structural, and other trust model document development requirements defined in Section 6.

In addition there are recommendations in Section 5 that organizations SHOULD consider when creating or maintaining trust model documents.

5. Algorithms and Parameters

Since [XMLDSIG] does not require support for all of the signature and hash algorithms needed for the trust model, this section adds requirements for supporting the RSA Algorithm signature method with SHA-256 algorithm and the ECDSAwithSHA256 signature algorithm. This section adds these selected algorithms into the trust model consistent with both RFC 4051 [RFC4051] and the currently under development [XMLDSIG-11]. The RSA algorithm refers to the RSASSA-PKCS1-v1_5 algorithm described in section 8.2 of RFC 3447 [PKCS1].

Other algorithms not otherwise required by [XMLDSIG] or this section MAY OPTIONALLY be used by content authors and supported by content consumers, but only the algorithms and parameters required by [XMLDSIG] and this section are assured to be interoperable across all implementations. If an algorithm identifier has been specified in [RFC4051], the identifier specified within [RFC4051] SHOULD be used. Section 7 includes additional processing requirements for content consumers.

NIST Federal Information Processing Standards (FIPS) 186-3, *Digital Signature Standard (DSS)* [FIPS186-3] and NIST Special Publication (SP) 800-57, *Recommendation for Key Management – Part 1: General* [SP800-57] provide additional information relating to security considerations in key size choice for various algorithms.

5.1 RSA-SHA256

The RSA Algorithm signature method with SHA-256 algorithm MUST be supported. Consistent with section 2.3.2 of [RFC4051] and section 6.4.2 of [XMLDSIG-11], the RSA Algorithm signature method with SHA-256 algorithm MUST be identified using the following algorithm identifier:

```
http://www.w3.org/2001/04/xmldsig-more#rsa-sha256
```

The `<dsig:SignatureValue>` content for this identifier MUST be the base64 encoding, as described in RFC 2045 [RFC2045], of the octet string, S, specified in section 8.2.1 of RFC 3447 [PKCS1]. Signature computation and verification does not require implementation of an ASN.1 parser. For the RSA Algorithm, content consumers MUST support 2048-bit keys and SHOULD support 3072-bit keys. Content authors SHOULD use a key size of either 2048 or 3072 bits.

5.2 ECDSA-SHA256

The ECDSAwithSHA256 signature algorithm MUST be supported, which is ECDSA [FIPS186-3] over the P-256 prime curve specified in Appendix D of [FIPS186-3] and using the SHA-256 algorithm. Consistent with section 2.3.6 of [RFC4051] and section 6.4.3 of [XMLDSIG-11], the ECDSAwithSHA256 MUST be identified using the following algorithm identifier:

```
http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256
```

The ECDSA algorithm signature is a pair of integers referred to as (r, s). The `<dsig:SignatureValue>` consists of the base64 [RFC2045] encoding of the concatenation of two octet-streams that respectively result from the octet-encoding of the values r and s, in that order. Integer to octet-stream conversion MUST be done according to the I2OSP operation defined in Section 4.1 of RFC 3447 [PKCS1] with the xLen parameter equal to the size of the base point order of the curve in bytes (32 for the P-256 curve).

5.3 Digest Algorithms

While content consumers are still REQUIRED to support the SHA-1 Digest algorithm as defined in section 6.2.1 of [XMLDSIG], content authors SHOULD NOT use the SHA-1 Digest algorithm. Content authors SHOULD instead use one of the algorithms defined within this section. The identifiers used below are consistent with either [RFC4051] or the identifiers used in *XML Encryption Syntax and Processing* [XMLENC], and with the current work occurring on [XMLDSIG-11]. The SHA-256 Digest algorithm MUST be supported by conforming implementations. SHA-384 and SHA-512 are OPTIONAL to support.

5.3.1 SHA-256

The SHA-256 algorithm [FIPS180-3] MUST be identified using the following algorithm identifier:

```
http://www.w3.org/2001/04/xmlenc#sha256
```

The SHA-256 algorithm produces a 256-bit digest string. The content of the `<dsig:DigestValue>` MUST be the base64 [RFC2045] encoding of the digest string viewed as a 32-octet octet stream.

5.3.2 SHA-384

The SHA-384 algorithm [FIPS180-3] MUST be identified using the following algorithm identifier:

```
http://www.w3.org/2001/04/xmldsig-more#sha384
```

The SHA-384 algorithm produces a 384-bit digest string. The content of the `<dsig:DigestValue>` MUST be the base64 [RFC2045] encoding of the digest string viewed as a 48-octet octet stream.

5.3.3 SHA-512

The SHA-512 algorithm [FIPS180-3] MUST be identified using the following algorithm identifier:

```
http://www.w3.org/2001/04/xmlenc#sha512
```

The SHA-512 algorithm produces a 512-bit digest string. The content of the `<dsig:DigestValue>` MUST be the base64 [RFC2045] encoding of the digest string viewed as a 64-octet octet stream.

6. Model Overview

The syntax and processing of the trust model is based on the [XMLDSIG] W3C Recommendation, and content authors and consumers MUST follow the conformance requirements found in [XMLDSIG]. This section provides a high-level overview and gives recommendations on how [XMLDSIG] can be used to establish a mechanism where signature information can be provided for the XML documents used within the security automation domain.

Figure 1 shows an informative, high-level composition of a signature. Not all signatures will contain all elements, and some signatures could contain additional elements. Content authors can create the signature block based on the documents necessary for their use case. Content consumers can validate the signature block prior to processing the signed content.

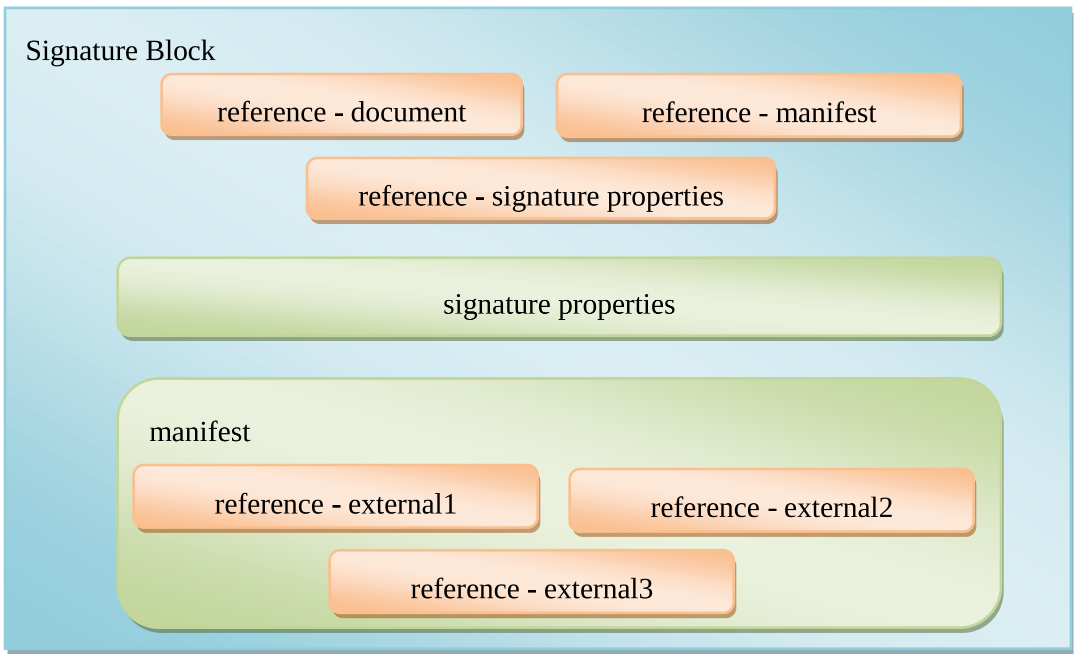

Figure 1 – High-Level Signature Diagram

6.1 Signature Types

As defined by [XMLDSIG], there are three main ways that a signature can relate to a given reference, and it is possible that the same signature will contain references with different signature relationships. The three possible signature relationships are:

- Detached - the signature is over content external to the signature itself
- Enveloped - the signature is embedded within the content that is signed
- Enveloping - the signature contains the content that is signed

The following subsections provide more information on selecting the appropriate style of signature.

6.1.1 Detached

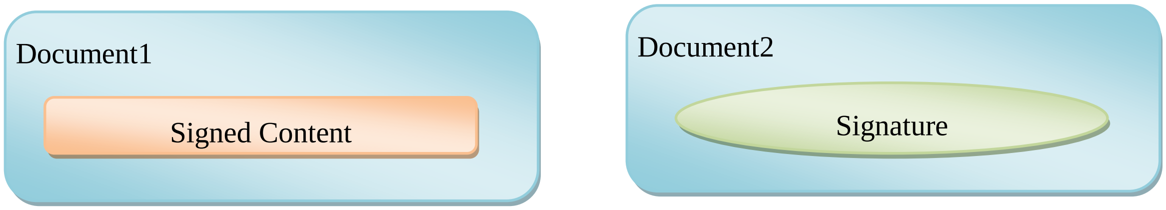

Figure 2 – Detached Signature in a Separate Document

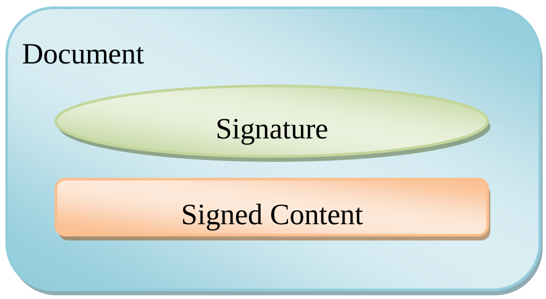

Figure 3 – Detached Signature in the Same Document

A detached signature typically occurs when the signature and signed content are separate. Figure 2 represents the case when the signed content and the signature are in two separate documents. Figure 3 represents a detached signature where the signed content and signature are in the same document but are sibling nodes (or a child node of a sibling). Note that in Figure 3 the “Signature” can occur either before or after the “Signed Content”. The consequence of a detached signature is that the content being signed may be managed independently, and it is not necessary for the content being signed to provide an element for containing the signature. It is necessary that another file containing the signature, or a file format capable of containing the signature and the signed content must be created or used. “Detached” is most commonly useful when a collection of documents must be signed with a single signature, or if a document must be signed but a signature element has not been provided.

6.1.2 Enveloped

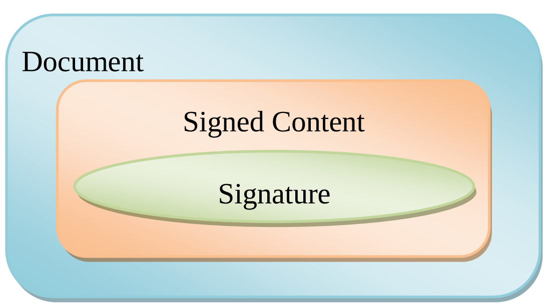

Figure 4 – Enveloped Signature

Figure 4 shows how an enveloped signature relates to the signed content. The signed content has an element that contains the signature. A named transform is used to exclude the signature element during signature validation. In contrast to the detached signature, when the signature is enveloped in the content being signed, a specific version of the signature specification must be referenced by the content being signed. Additionally, whenever content is signed, the signature will always be available with the content, unlike with a detached signature where the signature may be located separately. Enveloped is most commonly useful when a single standalone document must be signed independent of any other documents.

6.1.3 Enveloping

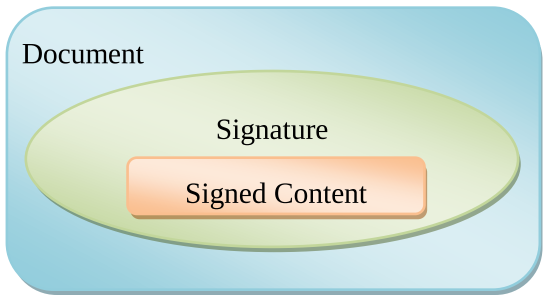

Figure 5 – Enveloping Signature

Figure 5 shows how an enveloping signature relates to the signed content. The signed content is contained as a child of the <dsig:Object> node within the signature. To process the signed content, the signature syntax will also need to be processed. If the same content is unsigned, it will have a different format from the signed version of the content. As with enveloped, the signature will always be available with the content if it has been signed. Most commonly, enveloping is useful for when the content is another signature that must signed. Manifest and signature properties also have an enveloping relationship to the signature which includes these elements.

6.2 XML Signature Syntax Overview

All signature content MUST conform to the [XMLDSIG] specification and validate against the schema found at http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-schema.xsd. Section 2.0 of [XMLDSIG] has a figure showing an informal representation of the syntax. Figure 6 is a modified version of that figure to show additional areas of interest. The additional highlighted items are the <dsig:KeyValue>, <dsig:X509Data>, <dsig:Manifest>, and <dsig:SignatureProperties> elements. The <dsig:KeyValue> and <dsig:X509Data> elements are ways to obtain the public key that can be used to validate the signature. In Figure 6 the "?", "+", and "*" characters represent the number of times the preceding element or attribute is to be used. "?" represents once or not at all, "+" represents one or more times, and "*" represents zero or more times.

```
<Signature ID?>
  <SignedInfo>
    <CanonicalizationMethod/>
    <SignatureMethod/>
    <Reference URI? >
      <Transforms/>?
      <DigestMethod/>
      <DigestValue/>
    </Reference>)+
  </SignedInfo>
  <SignatureValue/>
  <KeyInfo>
    <KeyValue/>?
    <X509Data/>?
  </KeyInfo>
  <Object ID>
    <Manifest>
      <Reference URI? >
        <Transforms/>?
        <DigestMethod/>
        <DigestValue/>
      </Reference>)+
    </Manifest>
  </Object>?
  <Object ID>
   <SignatureProperties>
      <SignatureProperty/>+
   </SignatureProperties>
  </Object>
  <Object ID?>*
</Signature>
```

Figure 6 – XML Signature Syntax Element Hierarchy

The `<dsig:Manifest>` element is used to provide additional references which compose the content. The `<dsig:SignatureProperties>` element is used to provide metadata about the signature. An additional use would be for the inclusion of timestamp information according to the recommendations in NIST SP 800-102, *Recommendation for Digital Signature Timeliness* [SP800-102].

Once a signature has been created, the signature and the content referred to by `<dsig:Reference>` elements cannot be reformatted, except as is permissible by the XML Canonicalization transform that has been applied [XML-C14N, XML-C14N11, and XML-exc-C14N]. The possible scope of reformatting is very limited and content consumers SHOULD maintain the format of received content.

6.2.1 SignedInfo

`<dsig:SignedInfo>` includes the canonicalization method for the signature block itself, the signature method, and references to the content that is part of what is signed. Any element outside of the `<dsig:SignedInfo>` element that is not referenced is not included as part of the signature validation.

According to [XMLDSIG] a `<dsig:SignedInfo>` element MUST include at least one `<dsig:Reference>`. If only one `<dsig:Reference>` is provided, it SHOULD be to the content being signed. An additional <dsig:Reference> to a <dsig:SignatureProperties> element as described in Section 6.2.3.2 SHOULD also be included. If the content being signed is dependent upon additional references, see Section 6.2.3.1 for additional guidelines.

6.2.2 KeyInfo

The <dsig:KeyInfo> element MAY be used to provide information about how to obtain the key needed for signature validation. In addition to the requirements in section 4.4 of [XMLDSIG], applications MUST implement support for the <dsig:X509Data> element in section 4.4.4 of [XMLDSIG].

RFC 4050 [RFC4050] describes a possible <dsig:KeyValue> representation for an ECDSA key. The representation and processing instructions described in [RFC4050] are not completely compatible with [XMLDSIG-11]; therefore, ECDSA keys SHOULD NOT be provided through a <dsig:KeyValue> element.

Note that unless a <dsig:Reference> to the <dsig:KeyInfo> is included, the <dsig:KeyInfo> is not validated as part of the signature.

6.2.3 Object

The <dsig:Object> element holds data that can be referenced, usually for an enveloping signature. The <dsig:SignatureProperties> and <dsig:Manifest> elements are both children of <dsig:Object>.

6.2.3.1 Manifest

The <dsig:Manifest> element SHOULD be used when additional document references beyond the main document reference are necessary. This is typically the case when a collection of documents is needed to represent all of the necessary content or when a primary document has dependencies on content in additional documents. When the <dsig:Manifest> element is used, there MUST be a <dsig:Reference> within the <dsig:SignedInfo> element which references the <dsig:Manifest>. See Section 6.2.4 for the requirements on how the reference is accomplished. The content of the <dsig:Reference> elements MUST follow the requirements in Section 6.2.4. A <dsig:Reference> element included as a child of a <dsig:Manifest> will not be validated during signature validation.

6.2.3.2 SignatureProperty

A <dsig:SignatureProperties> element SHOULD be included on a signature as a child element of <dsig:Object>. The <dsig:SignatureProperties> element MUST contain at least one <dsig:SignatureProperty> element. The <dsig:SignatureProperty> element captures metadata information about the signature. If the RECOMMENDED *<dt:signature-info>* element is included, it MUST be included as the lone child of a <dsig:SignatureProperty> element included within a <dsig:SignatureProperties> element. This parent <dsig:SignatureProperty> element MUST include the *@Target* attribute populated with "#" + ID of the signature. Table 2 describes the *<dt:signature-info>* data model.

Table 2 – dt:signature-info

Element Name: dt:signature-info				
Definition	A root element capturing common metadata about an XML digital signature.			
Properties	**Name**	**Type**	**Count**	**Definition**
	dc:creator	literal – string	0-n	The person, organization, or tool that created the signature.
	dc:date	literal – dateTime	0-1	The date and time when the signature was created.
	nonce	literal – token	0-1	A token value. Possible uses include ordering of requests and preventing replay attacks.

An example of a <dsig:SignatureProperties> is included below:

```
<dsig:Object>
    <dsig:SignatureProperties Id="signature-prop-global-id1">
        <dsig:SignaturePropertyTarget="#digital-sig-gloabl-id1">
            <dt:signature-info>
                <dc:creator>John Smith</dc:creator>
                <dc:creator>ACME Inc</dc:creator>
                <dc:date>2011-07-01T00:00:00Z</dc:date>
                <dsig:nonce>04EED3035045C9E7</dsig:nonce>
            </dt:signature-info>
        </dsig:SignatureProperty>
    </dsig:SignatureProperties>
</dsig:Object>
```

The XML Schema for the <dt:signature-info> element is at http://scap.nist.gov/specifications/tmsad/#resource-1.0.

6.2.4 References

References are an essential part of an XML digital signature. This section contains requirements specific to the construction of references. These requirements apply to a <dsig:Reference> that is a child of either <dsig:SignedInfo> or <dsig:Manifest>.

If the document that contains the signature is referenced, it SHOULD be referenced by setting the @URI attribute on <dsig:Reference> to the empty string (i.e., @URI=""). When referencing items in the signature that have an attribute of type xs:ID such as <dsig:Object>, <dsig:Manifest>, or <dsig:SignatureProperties>, they SHOULD be referenced using a URI fragment (e.g., @URI="#referenceIdentifier").

When referencing a <dsig:Object>, <dsig:Manifest>, or <dsig:SignatureProperties> from a <dsig:Reference>, the @Type attribute MUST be specified, and it MUST contain http://www.w3.org/2000/09/xmldsig#Object, http://www.w3.org/2000/09/xmldsig#Manifest, or http://www.w3.org/2000/09/xmldsig#SignatureProperties, respectively.

When specifying XPath transforms, content authors SHOULD use only XPath Filter 2.0 [XPath Filter-2] , which is consistent with XML Digital Signature best practices [XMLDSIG-BEST]. Due to the more limited support of XPath 2.0, XPath transforms SHOULD use only XPath 1.0 [XPath] expressions.

When referencing the root node of an XML document, if an ID exists on the root node that is not of type `xs:ID`, then the reference SHOULD specify an [XPath Filter-2] transform targeting the root node by ID. For example, if the root node of a document is `<root-node id="root123">`, then the [XPath Filter-2] expression would be "`root-node[@id = "root123"]`" with a `@Filter` attribute value of "`intersect`". This approach is preferable because if the signed document is later included as a child node within another XML document, the signature can still be valid (unless there is an ID conflict).

Unnamed XSLT transforms SHOULD be avoided. Specifications requiring XSLT transform capabilities SHOULD create named XSLT transforms to avoid the issues with XSLT transforms identified in [XMLDSIG-BEST].

When specifying multiple transforms on a reference, the transforms SHOULD be specified in this order:

1. Enveloped Signature Transform (only when the signature is enveloped[1])
2. XPath Filter 2 Transforms (if applicable)
3. Named or XSLT Transforms (if applicable)
4. XML Canonicalization (only if the last transform outputs XML)

This ordering resulted from issues with an implementation of the [XMLDSIG] specification, when the enveloped signature transform was not the first transform. Additionally, because there is no guarantee that a Named or XSLT transform will result in XML, those transforms SHOULD come after the XPath Filter 2 transforms.

6.3 Conventions

This section contains additional conventions that apply to the creation of the signature.

6.3.1 Canonicalization

No additional support for canonicalization algorithms is necessary beyond what is specified in [XMLDSIG]. Content authors SHOULD use the Canonical XML 1.1 method [XML-C14N11].

6.3.2 Countersigning

Countersigning is the creation of a signature for content that has already been signed while maintaining the previous signature. Keeping the previous signature allows for provenance to be preserved over the content. A countersigner is signing the existing signature and not the content itself; therefore, the existing signature MUST validate successfully prior to countersigning. When countersigning an existing signature, content authors MUST include the original signature as a child to a `<dsig:Object>` element of the new signature and reference the `<dsig:Object>` within the new signature. The original signature MUST then be removed from the document and replaced with the new countersigning signature.

6.3.3 Id Values

`<dsig:Signature>`, `<dsig:SignatureProperties>`, `<dsig:Manifest>`, and `<dsig:Object>` each have an `@Id` attribute. The `@Id` attribute for these elements SHOULD be globally unique to permit document composition.

---

[1] http://www.w3.org/TR/xmldsig-core/#def-SignatureEnveloped

7. Processing Requirements

All implementations MUST implement the processing requirements specified in [XMLDSIG]. This section describes additional general processing requirements that implementations of the trust model MUST follow to correctly process the trust model.

7.1 Signature Identifiers

If an algorithm identifier has been specified in [RFC4051] and the identifier specified within [RFC4051] was used, implementations SHOULD follow any processing guidance associated with the identifier as specified within [RFC4051]. If, during validation of a signature, a content consumer encounters an algorithm or algorithm parameter that the content consumer does not support, an error MUST be issued. Algorithm parameters also include any implicit parameters such as the length in bits of the key.

7.2 Signature Verification

While not a requirement, when performing signature verification, implementations are encouraged to follow the relevant best practices in *XML Signature Best Practices* [XMLDSIG-BEST].

7.3 Manifest References

Although the content within a `<dsig:Manifest>` element is validated, the content for a `<dsig:Reference>` element that is a child of a `<dsig:Manifest>` element is not validated during signature validation. All content consumers that validate a signature MUST also validate a reference according to the reference validation requirements identified in section 3.2.1of [XMLDSIG].

7.4 KeyInfo

When processing a signature, if the `<dsig:KeyInfo>` element has not been provided, then a content consumer MUST either issue an error or provide a method for associating the content with a key that can be used to validate the signature.

7.5 Countersigning

When a signature (i.e., countersigning signature) countersigns another signature (i.e., countersigned signature) by including the countersigned signature as a child element to a `<dsig:Object>`, and the countersigned signature specifies the “Enveloped Signature Transform”[2] on one of its references, then special processing rules apply. Specifically, after validating the countersigning signature, the countersigning signature MUST be replaced in the XML content by the countersigned signature. If the “Enveloped Signature Transform” is not specified on any of the countersigned signature’s references, then the replace step MAY be skipped. Lastly, the countersigned signature MUST be validated. An error MUST be issued if a chain of signature references results in a cycle.

---

2 http://www.w3.org/TR/xmldsig-core/#sec-EnvelopedSignature

Appendix A—Example Usage

Example demonstrations of the information in this document can be found at http://scap.nist.gov/specifications/tmsad/#resource-1.0. Examples are:

- signing/hashing of a single document
- signing with a manifest
- countersigning (signing an already signed document)

Appendix B—References

B.1 Normative References

[FIPS180-3] United States. National Institute of Standards and Technology. Federal Information Processing Standards Publication 180-3, *Secure Hash Standard (SHS)*. October 2008. See http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf.

[FIPS186-3] United States. National Institute of Standards and Technology. Federal Information Processing Standards Publication 186-3, *Digital Signature Standard (DSS)*. June 2009. See http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf.

[PKCS1] Jonsson, J. and B. Kaliski (2003). *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*. February 2003. See http://www.ietf.org/rfc/rfc3447.txt.

[RFC2045] Freed, N. and N. Borenstein, (1996). *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. November 1996. See http://www.ietf.org/rfc/rfc2045.txt.

[RFC2119] Bradner, S. (1997). *Key words for use in RFCs to Indicate Requirement Levels*. March 1997. See http://www.ietf.org/rfc/rfc2119.txt.

[RFC4051] Eastlake, D. (2005). *Additional XML Security Uniform Resource Identifiers (URIs)*. April 2005. See http://www.ietf.org/rfc/rfc4051.txt.

[XML-C14N] Boyer, John (2001). *Canonical XML Version 1.0*, W3C Recommendation, March 2001. See http://www.w3.org/TR/2001/REC-xml-c14n-20010315 or http://www.ietf.org/rfc/rfc3076.txt.

[XML-C14N11] Boyer, John and Glenn Marcy (2008). *Canonical XML Version 1.1*, W3C Recommendation, May 2008. See http://www.w3.org/TR/2008/REC-xml-c14n11-20080502.

[XMLDSIG] Eastlake, Donald, et al. (2008). *XML Signature Syntax and Processing, 2nd Edition*, W3C Recommendation, June 2008. See http://www.w3.org/TR/xmldsig-core/.

[XML-exc-C14N] Boyer, John, Donald Eastlake, and Joseph Reagle (2002). *Exclusive XML Canonicalization Version 1.0*, W3C Recommendation, July 2002. See http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/.

[XPath] Clark, James and Steve DeRose (1999). *XML Path Language (XPath) Version 1.0*, W3C Recommendation. October 1999. See http://www.w3.org/TR/1999/REC-xpath-19991116.

[XPath Filter-2] Boyer, John, Merlin Hughes, and Joseph Reagle (2002). *XML-Signature XPath Filter 2.0*, W3C Recommendation, November 2002. See http://www.w3.org/TR/2002/REC-xmldsig-filter2-20021108/.

B.2 Informative References

[FIPS140-2] United States. National Institute of Standards and Technology. Federal Information Processing Standards Publication 140-2, *Security Requirements for Cryptographic Modules*. 25 May 2001. See http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf.

[RFC4050] Blake-Wilson, S., et al. (2005). *Using the Elliptic Curve Signature Algorithm (ECDSA) for*

XML Digital Signatures. April 2005. See http://www.ietf.org/rfc/rfc4050.txt.

[SP800-57] United States. National Institute of Standards and Technology. Special Publication 800-57, *Recommendation for Key Management – Part 1: General*. March 2007. See http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf.

[SP800-102] United States. National Institute of Standards and Technology. Special Publication 800-102, *Recommendation for Digital Signature Timeliness*. September 2009. See http://csrc.nist.gov/publications/nistpubs/800-102/sp800-102.pdf.

[XML] Bray, Tim, et al. (2008). *Extensible Markup Language (XML) 1.0, 5th Edition*, W3C Recommendation, November 2008. See http://www.w3.org/TR/2008/REC-xml-20081126/.

[XMLDSIG-11] Eastlake, Donald, et al. (2011). *XML Signature Syntax and Processing Version 1.1*, W3C Candidate Recommendation, March 2011. See http://www.w3.org/TR/2011/CR-xmldsig-core1-20110303/.

[XMLDSIG-BEST] Hirsch, Frederick, and Datta, Pratik. (2010). *XML Signature Best Practices*, August 2010. See http://www.w3.org/TR/xmldsig-bestpractices/.

[XMLENC] Eastlake, Donald and Joseph Reagle. (2002). *XML Encryption Syntax and Processing*, W3C Recommendation, December 2002. See http://www.w3.org/TR/xmlenc-core/.

Appendix C—Change Log

Release 0 – July 2011

- Initial public release